Chevelle Boyer

SE 201

Assignment 1

28 September 2019

The first thing I did for this assignment was load song lyrics into a string. I did this for two reasons, I wanted to start with a small sample of words as practice, but also because the actual assignment wasn’t yet available. I created a txt file with four stop words: the, it, and, because. The first thing that came to mind was loop through each word in my array of words. I did this to ensure the array contained the words in the right format. It didn’t. I forgot to parse out the punctuation and normalize the capitalization. After doing those things it was looping and printing correctly. I added an if statement as the first condition in the for loop to check whether the word was a stop word or not. I knew I didn’t want to use any conditionals to check if the word was already counted and stored or not. For that reason, I decided to use a dictionary to hold the words and their corresponding frequencies. I knew this would work because dictionary keys must be unique, so all I had to do was add the key to the dictionary and increment the current value. When I ran it to test this, however, it failed with key errors. This was caused when inserting new words that were not yet in the dictionary. The way I was inserting key value pairs was assuming the key was already in there. To get around this I encased the insertion in a try except. In the try I assumed the key was already there and incremented its current value by one. If it failed with a key error exception, I added the key with a value of one. I really wanted to do this without using a loop to iterate through all the words. To do this I added each word into a queue as I read the file. Then instead of a using a for loop to check for conditionals and add to the dictionary, I used while Queue() not empty and dequeued each word. I thought this seemed correct. After listening to you and more examination I realized it was still iterating through each item and offered no advantage since the order of the words didn’t matter. I scraped that idea.

I’m embarrassed to admit, but the sorting took more than a couple of tries. The first thing I tried was looping through the key and values, then appending a tuple containing the value as the 0th element and the key as the 1th element to a bigger list. This is confusing but basically it looked like this:

**[ [value, key], [value, key], [value, key]….]**

By the time I was done I thought now it would be easy enough to iterate and sort this loop. I passed this list to python’s sorted() function and used the optional “key” parameter to tell it to sort by the 0th element of each sub list. This worked okay but it felt really hacky and unnecessary. All of the sudden I had this AWESOME idea. I thought I was so clever. The idea was to scratch the dimensional array and just iterate through the dictionary key and values. Then I would insert the (key, value) tuple into the new array at the index of whatever the value was!!! So basically, if the value of the tuple was 2, I inserted the tuple into the 2th position. If there was another value with 2 in the array that’s okay because everything would just shift over and stay sorted. I thought I was the smartest person in the world at this point. It was instantly sorted in one loop, I thought. Then I quickly realized that as soon as there was two of the same values, it unsorted everything. Here’s why: At first it works, let’s say the first 3 values are 2, 6, and 4 the list looks like:

**[ Null, Null, (“word”, 2), Null, (“word”, 4), Null, (“word”, 6) ]**

And maybe the next two values are 4 again:

**[ Null, Null, (“word”, 2), Null, (“word”, 4), (“word”, 4), (“word”, 4), Null, (“word”, 6) ]**

So far still sorted so what’s the problem? If the next value is 5:

**[ Null, Null, (“word”, 2), Null, (“word”, 4), (“word”, 5), (“word”, 4), (“word”, 4), Null, (“word”, 6) ]**

Well great. Next I thought, well maybe I can use pythons sorted function and make the “key” parameter choose the value of the dictionary item instead of the key, to sort. This worked! What I did was create a list and set it equal to the sorted items. Like this:

**frequentWords = sorted( frequencies.iteritems(), key = lambda x : x[1] )**

The lambda is like a nameless function. It would be the same as if I defined and used a function elsewhere that returned the 1th element from whatever x was. But I guess I don’t need to explain that to you. Now that it works properly there is still so much about it that I hate. I feel as though I’m using too many conditionals. I have a global counter to control when I have the first 25 words, which feels….icky. I am also using pythons replace() 3 separate times right after another to clean up and normalize the words. I tried using a regex statement instead of the replace() function but after timing them against each other the replace() was much faster than the regex. I know this has to be because the logic of my regex statement is poorly written. It feels wrong, nonetheless. Sorry for the mouthful, hopefully you followed all of that.